Реализация базового алгоритма восстановления разделённых грамматик

К.В. Савельев

***Аннотация* — Работа посвящена программной реализации базового алгоритма восстановления грамматики по заданному множеству предложений, разработанного проф. С.Ю. Соловьёвым. Надежность и работоспособность созданного программного комплекса продемонстрирована на примере восстановления всевозможных допустимых грамматик для заданного реального положительного образца. Описывается разработанное оригинальное программное обеспечение для интерактивного сопровождения и графической поддержки решения задачи восстановления грамматик.**

***Ключевые слова* — восстановление грамматик; префиксные сети; разделённые грамматики.**

I. Введение

Проблема восстановления грамматики по заданному образцу - это задача определения совокупности правил построения предложений, которые допускают воспроизведение предложений заданного образца. Базовый алгоритм восстановления разделённых грамматик [1], рассматриваемый в данной статье, имеет параметризацию, позволяющую получить множество восстановленных грамматик по каждому заданному образцу. В дальнейшем, на основе различных критериев среди этого множества можно выбрать единственную грамматику, которая поступит в распоряжение исследователя. Критерии отбора могут быть различным. [9], [10]. Например, можно отбирать ту грамматику из получившегося множества, которая позволяет сконструировать максимальное количество различных типов предложений, не содержащихся в заданном образце. Другим критерием может быть хорошее соответствие заданной грамматики, с грамматиками, восстановленными по другим образцам [11]. В заданном контексте под «образцом» понимается, так называемый, положительный образец, то есть восстановленная по нему грамматика, должна строить все предложения заданного образца.

Общая модель грамматического вывода представляет собой взаимодействие двух объектов - *источника* и *анализатора*. Источником является генератор предложений, то есть совокупностей элементов алфавита (символов языка), являющихся элементами *языка* источника.

Савельев Кирилл Владимирович., магистр 2 курса, факультет ВМК МГУ имени М.В. Ломоносова, e-mail: msuait@bk.ru. Работа представляет собой результат магистерской диссертации

Язык источника - это набор предложений, которые могут быть использованы для построения образцов. При построении грамматики используются свои символы, которые делятся на терминалы, нетерминалы и два служебных символа. Терминалами являются символы алфавита языка источника, нетерминалы – это непересекающиеся с множеством терминалов символы, служащие для построения грамматических формул. Служебные символы используются при записи этих формул. Анализатор - это алгоритм, который обрабатывает совокупность предложений, сгенерированных источником (заданный образец), и по этому образцу формирует искомую грамматику, то есть, четвёрку множеств $G = \langle N, \Sigma, P, S \rangle$, где

N - конечное множество нетерминальных символов (нетерминалов),

Σ - непересекающееся с N конечное множество терминальных символов,

P - конечное множество правил,

S - выделенный из N символ, называемый начальным [2], [3].

Вообще говоря, задача восстановления грамматики произвольным методом по заданному образцу, может быть неразрешима для случая языка источника бесконечной мощности, так как порождаемая им грамматика может содержать бесконечное число правил.

В случае конечного языка, задача является всегда разрешимой, так как может сводиться к прямому отображению каждого конкретного предложения в правило, которое это предложение порождает.

В данной статье рассматривается возможность восстановления грамматики по положительному образцу для языка бесконечной мощности с конечным множеством правил [1].

II. Структура базового алгоритма и операции.

Структура базового алгоритма состоит из четырёх этапов:

1. Формирование начальной π-сети.
2. Преобразование начальной π-сети.
3. Построение грамматики, порождаемой данной π-сетью.
4. Проверка качества полученной грамматики.

Этап формирования начальной π-сети состоит в построении первичной сети, анализом вершин графа, разбиением множества этих вершин на классы эквивалентности и проведения операции совмещения.

Формирование первичной π-сети состоит в том, что строится конечное множество сетей (цепочек символов),

каждая из которых представляет собой префиксную π-сеть, полученную путём прямого отображения каждого предложения из положительного образца. В результате получается n (n – количество предложений в образце) несвязанных между собой линейных π-сетей. Каждая π-сеть состоит из вершин, с которыми ассоциируются конкретные символы заданного предложения и связывающих их рёбер. Граф начальной π-сети состоит из объединения линейных π-сетей. Далее следует разбиение вершин на классы эквивалентности.

Вершины принадлежат одному классу эквивалентности, если:

- Входные в эти вершины рёбра помечены одним и тем же символом.
- Родители этих вершин принадлежат одному классу эквивалентности.
- Все входные вершины цепочек имеют один класс эквивалентности, равно как все выходные вершины также принадлежат одному классу эквивалентности.

На рис. 1 представлено разбиение вершин графа по классам эквивалентности. Вершины, принадлежащие одному классу эквивалентности, отмечены на рисунке одним и тем же цветом.

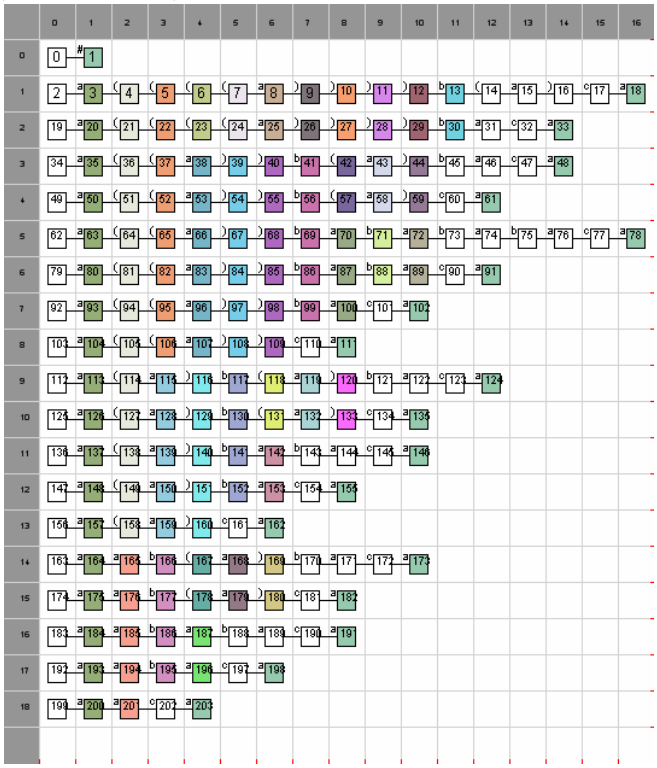

Рис. 1. Классы эквивалентности.

Вершины одного класса эквивалентности объединяются в одну посредством операции совмещения. Операция совмещения позволяет построить из первичной префиксной π-сети новую двухполюсную сеть [6]. Эквивалентными вершинами сети являются:

- входные полюса начальной сети,
- выходные полюса начальной сети,
- равно-достижимые внутренние вершины сети.

Две внутренние вершины A и B являются равно-достижимыми, если в соответствующих исходных сетях найдутся рёбра (C, A) и (D, B), такие что они помечены одним и тем же символом и вершины C и D принадлежат к одному классу эквивалентности. После завершения операции совмещения, первичная сеть переходит в начальную сеть (рис. 2). [5]

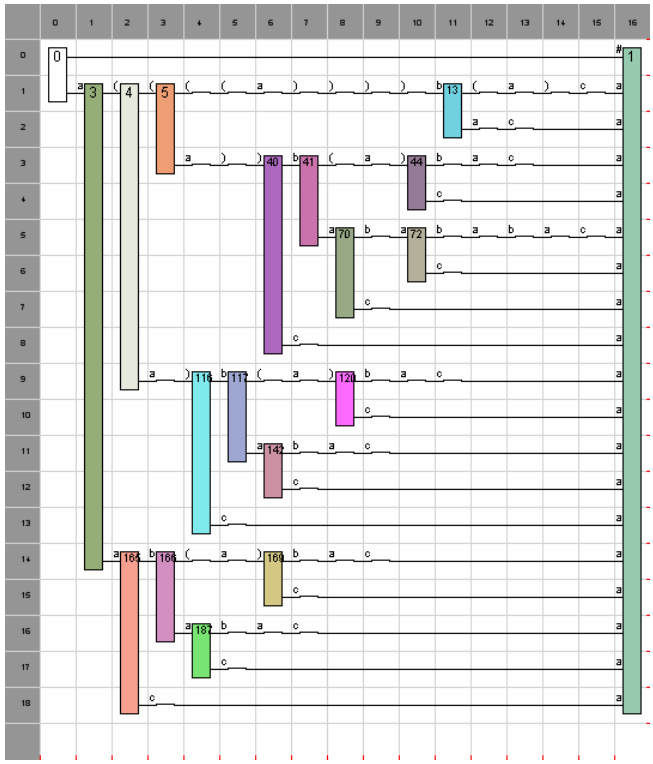

Рис. 2. Совмещённая сеть.

Начальная сеть трансформируется в окончательную сеть с помощью последовательного многократного применения операций правой факторизации и правого деления.

Операция правой факторизации [7] применяется к совмещённой сети и заключается в формировании выходных вершин подсетей. Рассматриваются все вершины, полученные на этапе совмещения, которые могут являться входными вершинами подсетей и для каждой такой подсети ищется её выходная вершина. Найдя эту выходную вершину, убеждаемся в том, что все входящие рёбра помечены одним и тем же символом. Если источники этих рёбер представляют собой линейные вершины, то эти линейные вершины можно объединить в одну вершину с сохранением всех входящих ребёр. При этом, полученная вершина является новой выходной вершиной для рассматриваемой подсети. Как и операция совмещения, данная операция упрощает сеть, уменьшая количество вершин в сети.

Формальное описание операции правой факторизации:

Если в данной сети с выходным полюсом v

1. все ребра $(v1, v)$, …, (vn, v) $(где\ n > 1)$ помечены одним и тем же символом b,
2. каждая вершина $vi (i = 1..n)$ имеет ровно одно

входящее ребро (wi, vi),

то в результирующей s-сети:

1. Ребра $(w2, v2)$, ... , (wn, vn) следует заменить (с сохранением меток) на ребра $(w2, v1)$, ... , $(wn, v1)$ соответственно.
2. Ребра $(v2, v)$, … , (vn, v) следует удалить.

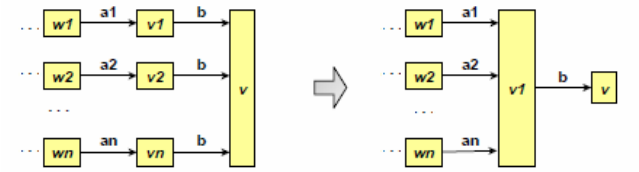

Рис. 3. Пример операции правой факторизации

Ниже представлен пример операции правой факторизации для некоторого положительного образца (рис. 4).

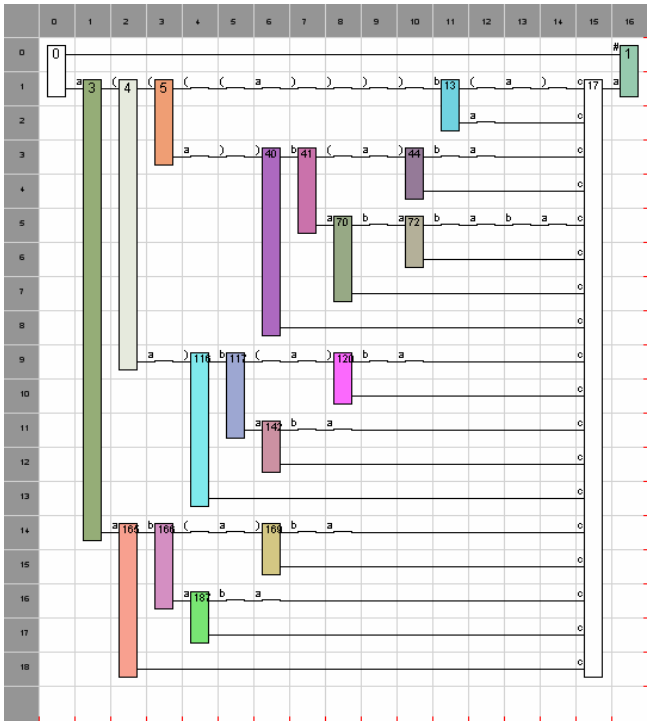

Рис. 4. Правая факторизация

Суть операции правого деления сводится к выделению таких подсетей, которые порождают одинаковые или почти одинаковые (вложенные) множества предложений, с последующим совмещением таких подсетей в одну подсеть.

Первым шагом этой операции является построение иерархии подсетей, отвечающих данному состоянию общего графа. На рис. 5 представлена иерархия подсетей, отвечающая некоторой трансформации начальной сети, изображённой на рис. 4.

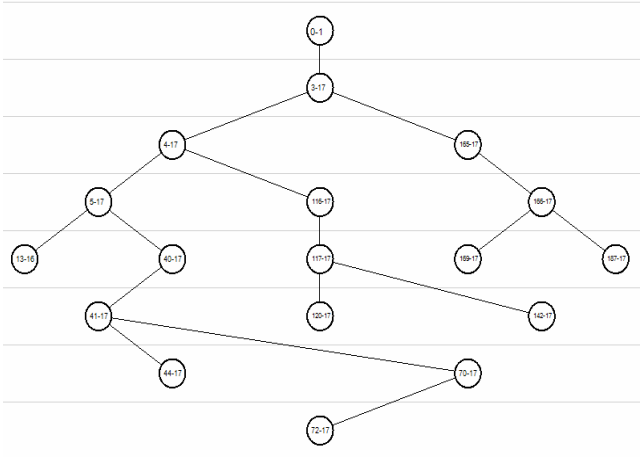

Рис. 5. Иерархия подсетей

Эта иерархия позволяет отбирать подсети и проверять их на возможность объединения в единую подсеть, в чём и состоит суть операции правого деления.

В операции правого деления участвуют оба параметра t и h. Параметр t определяет ограничение сверху на длину предложений языка подсети, которые будут участвовать при сравнении его с языками других подсетей. Параметр h используется в алгоритме следующим образом: для каждой из всех подсетей, которые проверяются на возможность объединения, вычисляется глубина расположения начальной вершины этой подсети от вершины ветвления графа иерархии. Если эта глубина меньше h, то такая подсеть допускается к операции правого деления, иначе, подсеть удаляется из графа сети, включая линейную цепочку, отделяющую её от вершины ветвления. После каждого изменения структуры сети необходимо перестроить иерархию подсетей и повторить операцию правого деления сначала. После завершения этого процесса проводится анализ возможности проведения операции правого деления.

Сравниваются локальные языки, отвечающие каждой из рассматриваемых подсетей. Если языки совпадают или существует с поглощающим языком, то подсети объединяются. Операция правого деления завершена.

Формальное описание операции правого деления:

Операция правого деления заданной р-сети

– преобразует (если это возможно) заданную р-сеть в s-сеть;

– использует проверку подобия р-сетей;

– зависит от дополнительного целочисленного параметра *h*.

Операция правого деления выполняется так (рис. 6):

если в заданной сети (с входным полюсом *w* и выходным полюсом *v*) существует набор вершин $v1$, … , vm, $vm+1$, … , vn $(m > 1)$ таких, что:

– любой маршрут проходит ровно через одну вершину $v1$, … , vn ;

– для $i = 1..m$:

(а) длина путей из *w* в *vi* не превосходит *h*, и

(б) все <*vi* ; *v*> являются попарно подобными р-подсетями,

а их совмещение есть р-сеть П;

– для $j = m+1..n$:

(а) длина путей из *w* в *vj* превосходит *h*, и

(б) все подсети $Aj = < vj; v >$ удовлетворяет условию:

$$\{x \in язык(A_j) \mid длина(x) \le t\} \subseteq \{y \in язык(\Pi) \mid длина(y) \le t\};$$

то результирующая s-сеть получается последовательным соединением левого фрагмента заданной сети и сети П.

Заметим, что при правом делении префиксная сеть, вообще говоря, меняется весьма существенно: часть предложений может «потеряться», а часть – появиться.

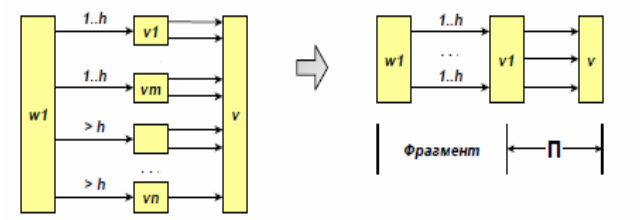

Рис. 6. Пример операции правого деления

В зависимости от значений параметров t и h, последовательность и характер трансформации начальной сети в окончательную может протекать по различным сценариям. Для того чтобы построить все возможные грамматики, отвечающие заданному образцу, нужно выполнить цикл трансформации при всех допустимых значениях t и h.

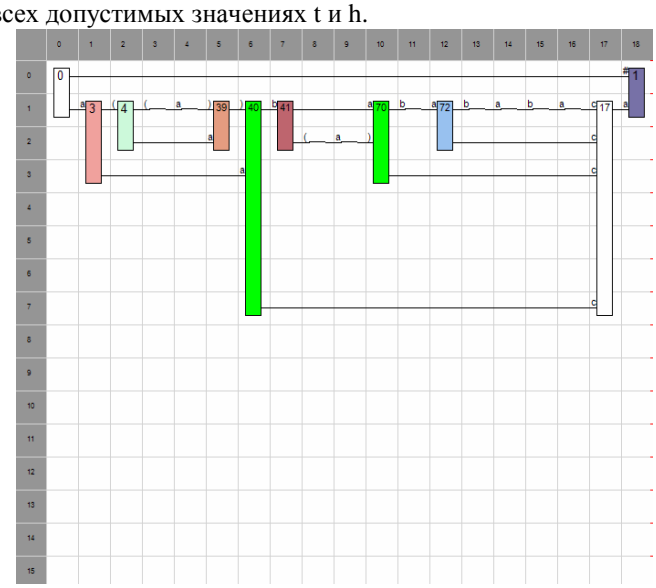

Рис. 7. Окончательный вид преобразованной сети

Описанные выше операции правой факторизации и правого деления применяются в базовом алгоритме многократно на каждой итерации циклов по t и h. В результате при фиксированных значениях t и h получается вид графа сети, который не может быть трансформирован с помощью указанных операций. На рис. 7 представлена окончательная сеть, полученная при некоторых t и h, к которой неприменима ни одна из вышеуказанных операций. Далее осуществляется попытка построения допустимой грамматики по данному виду сети.

Для построения грамматических правил вывода предложений по полученной структуре сети, необходимо выделить в этой сети классы подобия, составляющих её подсетей. Подсети из одного класса подобия порождают одно и то же правило вывода предложений. Две подсети являются подобными для заданного значения параметра t, если языки этих подсетей совпадают или язык одной подсети содержится в языке другой подсети. Стоит отметить, что в отличие от операции подобия подсетей на этапе трансформации, в данной операции при анализе языков подсетей не учитываются маршруты основной сети, выходящие из конечных вершин сравниваемых подсетей. Выделение классов подобия подсетей порождает множество уникальных грамматических правил вывода.

Таким образом, построение грамматики осуществляется по следующему алгоритму:

1. Разбить все подсети на классы подобия (эквивалентности) по вышеописанному критерию.
2. Каждому классу эквивалентности поставить в соответствие уникальный символ (нетерминал).
3. Создать пустую очередь подсетей, в которую добавить корень иерархии подсетей. Начать цикл обработки подсетей.
4. Выбрать очередную подсеть из очереди, построить грамматическое правило, отвечающее этой подсети, и поместить его во множество грамматических правил. Включить в конец очереди все подсети данной подсети, являющиеся её прямыми потомками. Если очередь пуста, то ***завершить*** работу алгоритма построения грамматики и выдать полученный набор правил (грамматику). Перейти в начало п. 4.

На рис. 8 изображена иерархия подсетей [8] соответствующая окончательной сети. Цветом выделены подобные подсети.

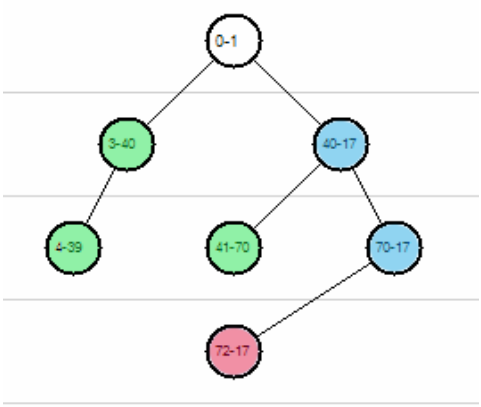

Рис. 8 Подобные подсети

Присвоение каждому классу уникального символа:
Белый: *(0-1) - S;*
Зелёный: *(3-40), (4-39), (41-70) - A;*
Синий: *(40-17), (70-17) - B;*
Красный: *(72-17) - C;*

Процедура восстановления начинается с корня иерархии подсетей - в данном случае с подсети (0-1). Восстановление правила из этой сети, с учётом прямых подсетей-потомков, даёт следующее правило:

$$S \Rightarrow \# \mid aABa$$

В очередь подсетей помещаются подсети-потомки. Далее, по алгоритму построения грамматических правил в результате анализа этих потомков, получим следующие правила:

$$A \Rightarrow a \mid (A)$$

$$B \Rightarrow c \mid bAB$$

В очередь подсетей добавляется последняя оставшаяся подсеть-потомок. Обработка этой подсети формирует правило:

$$C \Rightarrow c \mid babac$$

На этом построение грамматических правил закончено и сформирована следующая грамматика:

$$\begin{cases} S \Rightarrow \# \mid aABa \\ A \Rightarrow a \mid (A) \\ B \Rightarrow c \mid bAB \\ C \Rightarrow c \mid babac \end{cases}$$

Стоит отметить, что последнее правило $C \Rightarrow c \mid babac$ было сформулировано в соответствии с алгоритмом построения грамматических правил, однако, как следует из анализа грамматики в целом, правило C не имеет ссылок из других правил данной грамматики, то есть не используется в построении предложений. При этом, если данная грамматика является допустимой, то все предложения строящиеся с помощью этого правила, могут быть построены с использованием других правил данной грамматики. Такие избыточные правила, являющиеся следствием формализма алгоритма построения, исключаются из результирующей грамматики. Тем самым, окончательная грамматика, отвечающая данной структуре, имеет вид:

$$\begin{cases} S \Rightarrow \# \mid aABa \\ A \Rightarrow a \mid (A) \\ B \Rightarrow c \mid bAB \end{cases}$$

Построение грамматики завершено. Построенную грамматику теперь нужно проверить на допустимость. Под допустимыми грамматиками в этой статье понимаются такие грамматики, которые порождают заданный образец. Выбирается каждое предложение из исходного образца и производится попытка вывода этого предложения из полученной грамматики. При первой же неудачной попытке подобного рода, восстановленная грамматика считается недопустимой. Если все предложения исходного образца строятся успешно с помощью данной грамматики, то она является допустимой и включается в список грамматик, восстановленных по данному образцу.

Как отмечалось ранее, в базовом алгоритме осуществляется цикл восстановления грамматик при всех возможных способах трансформации исходного графа сети. После завершения построения и проверки данной грамматики происходит изменение значений параметров цикла t и h и базовый алгоритм переходит на следующую итерацию, осуществляя трансформацию исходного графа по новым правилам, определяемым значениями этих параметров, вплоть до построения и проверки следующей грамматики. После завершения работы цикла базовый алгоритм заканчивает работу, выдавая всё множество построенных допустимых грамматик с указанием для каждой грамматики соответствующих ей значений t и h.

III. Демонстрация работы базового алгоритма в реальном случае восстановления грамматик.

Проиллюстрируем все этапы работы базового алгоритма на реальном примере восстановления грамматик по следующему образцу:

 # aaca aabaca a(a)ca aab(a)ca aababaca a((a))ca a(a)baca aab(a)baca a((a))baca a(a)b(a)ca a(a)babaca a((a))b(a)ca a((a))babaca a(a)b(a)baca a((((a))))baca a((a))b(a)baca a((((a))))b(a)ca ***a((a))bababababaca***

Для данного реального примера допустимые значения параметров базового алгоритма варьируются в следующих диапазонах: $\begin{pmatrix} 1 \le t \le 16 \\ 1 \le h \le 16 \end{pmatrix}$, где ограничение сверху определяется длиной максимального по размеру предложения образца, выделенного жирным шрифтом. Вид начальной сети и последовательность её трансформаций проиллюстрирована рис. 9-15. По окончательному виду рис. 15 строится грамматика.

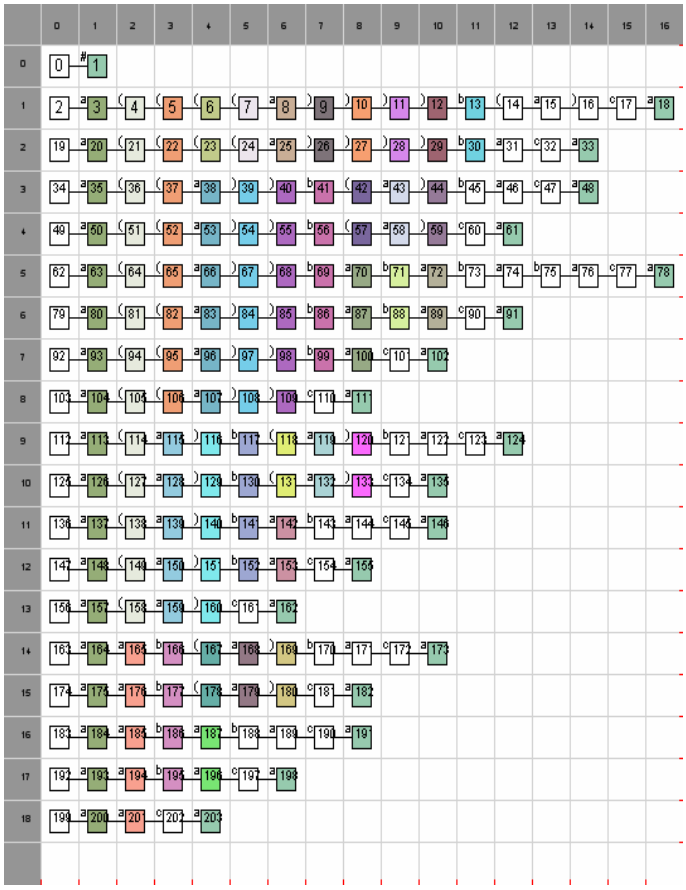

Рис. 9. Начальная сеть с выделенными классами эквивалентности.

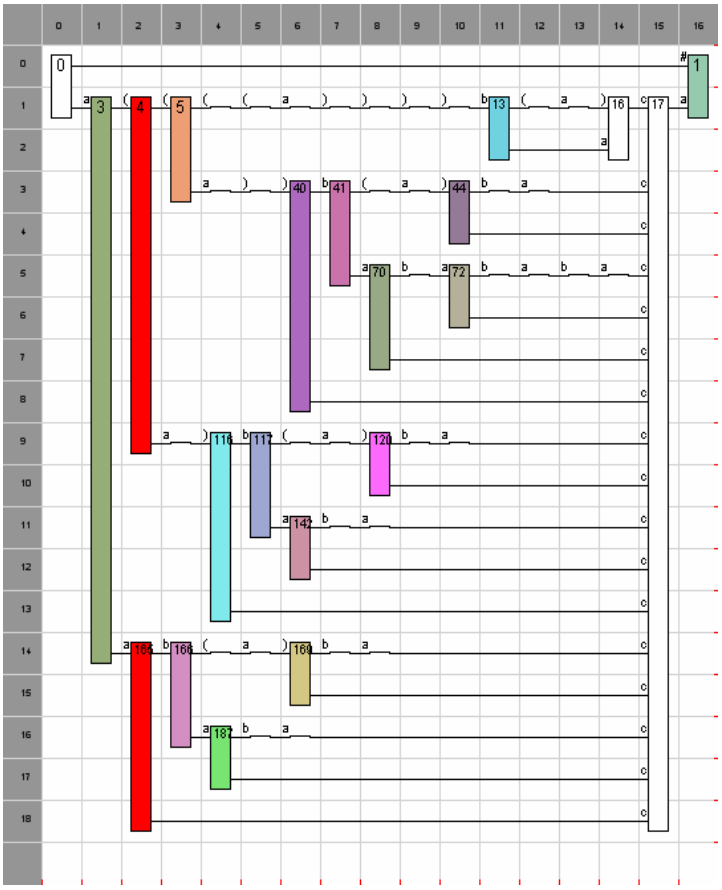

Рис. 11. Сеть после операций правой факторизации.

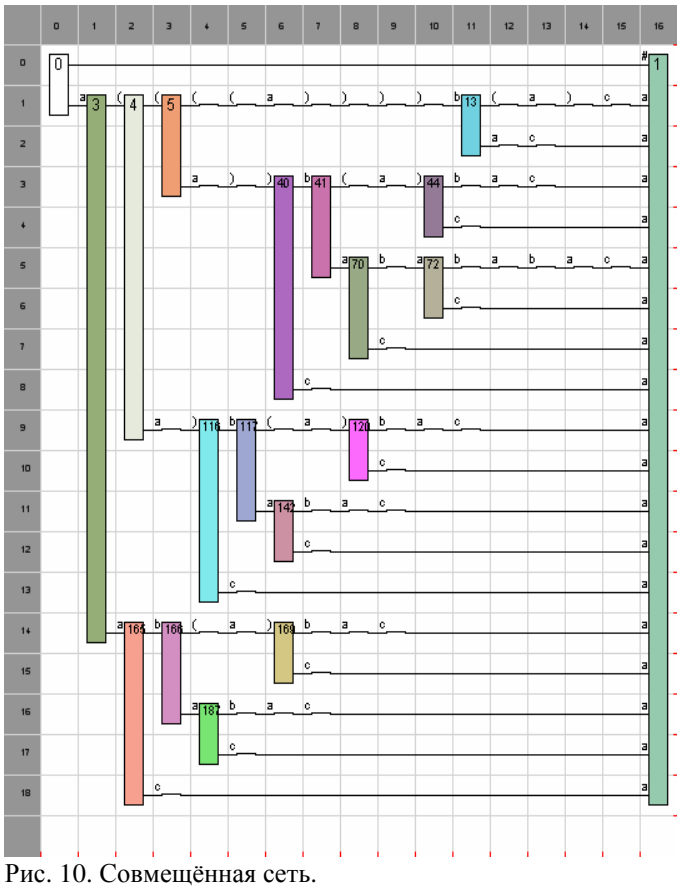

Рис. 10. Совмещённая сеть.

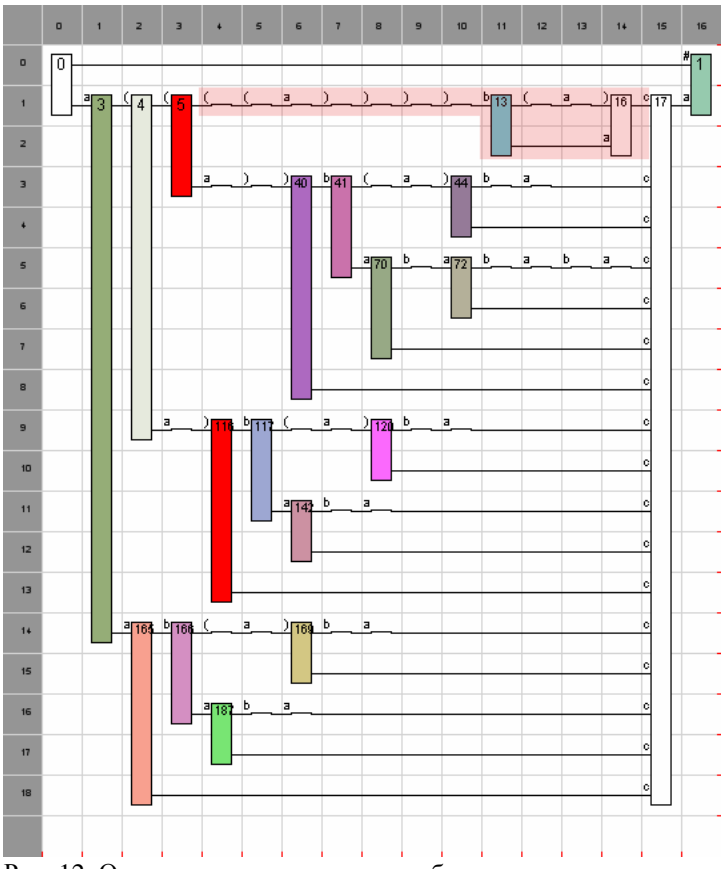

Рис. 12. Операция правого деления: обнаружение подсети на расстоянии > h.

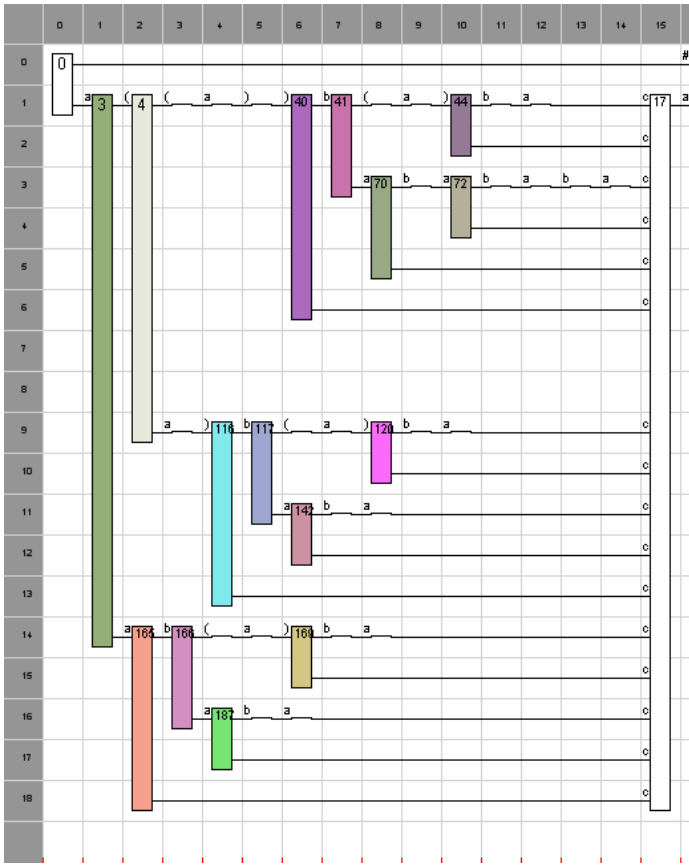

Рис. 13. Удаление подсети на расстоянии > h.

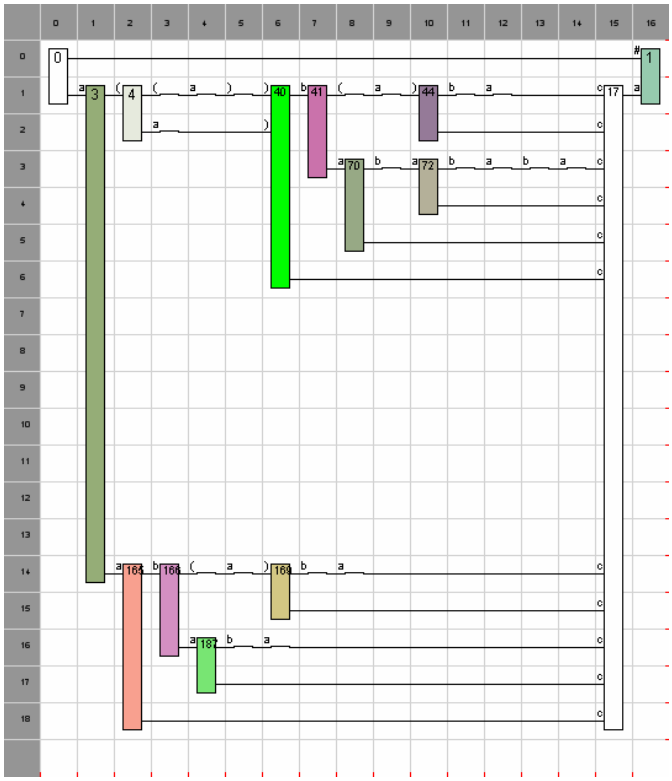

Рис. 14. Результат операции правого деления.

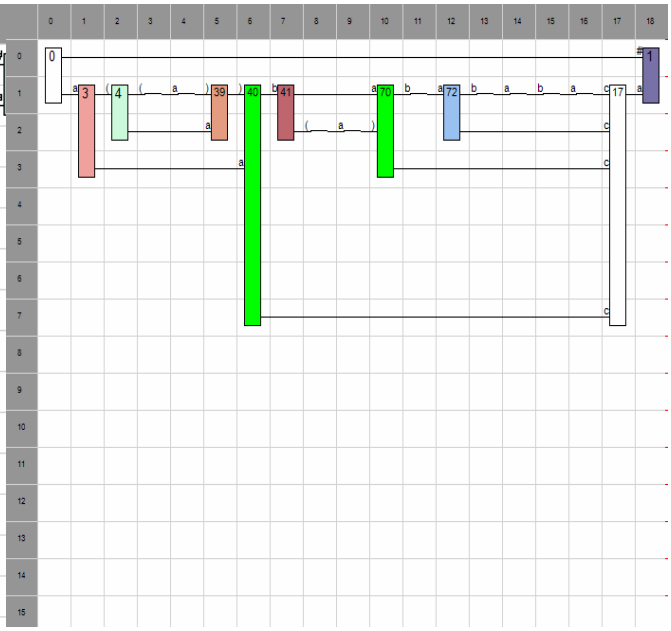

Рис. 15. Окончательный вид преобразованной сети.

Полная обработка заданного образца предполагает получение всевозможных грамматик, отвечающих данному образцу, что достигается тем, что исходная сеть, соответствующая данному образцу подвергается всевозможным допустимым трансформациям с использованием принятого в алгоритме набора операций.

Цепочка трансформаций исходной сети определяется значениями параметров алгоритма t и h. Для заданного образца значение этих натуральных параметров варьируется от 1 до $P_{\max}$, где $P_{\max}$ - максимальная длина цепочки символов, составляющих предложения данного образца. В рассматриваемом нами реальном примере значение $P_{\max}$ определяется длиной выделенного в образце слова, т. о. $P_{\max} = 16$.

Итак, для полной обработки образца, данный алгоритм должен совершить $16*16 = 256$ итераций, что соответствует изменению параметрам t и h в диапазонах: $\begin{pmatrix} 1 \le t \le 16 \\ 1 \le h \le 16 \end{pmatrix}$.

В результате завершения обработки образца на каждой из 256 итераций была построена допустимая грамматика, либо оказывалось, что допустимых грамматик при данном наборе t и h не существует. Всего было получено 6 различных грамматик [4], которые приведены ниже:

- S−>#|aAa
  A−>(B|aJ
  B−>(C|a)J
  C−>((a))))bDc|a))E
  D−>(a)|a
  E−>bF|c
  F−>(a)G|aH
  G−>bac|c
  H−>baI|c
  I−>babac|c
  J−>bDG|c

- S->#|aAa
  A->(B|aE
  B->(C|a)E
  C->((a))))bDc|a))E
  D->(a)|a
  E->bF|c
  F->(a)G|aH
  G->bac|c
  H->baI|c
  I->babac|c

- S->#|aAa
  A->(B|aE
  B->(C|a)E
  C->((a))))bDc|a))E
  D->(a)|a
  E->bDF|c
  F->baG|c
  G->babac|c

- S->#|aAa
  A->(B|aG
  B->(C|a)G
  C->((a))))bDc|a))G
  D->(a)|a
  G->bH|c
  H->(a)G|aG

- S−>#|aAa
  A−>(B|aF
  B−>(C|a)F
  C−>((a))))bDc|a))F
  D−>(a)|a
  F−>bDF|c

- S−>#|aCAa
  A−>bCA|c
  C−>(C)|a

- допустимые грамматики отсутствуют.

В целом, была получена следующая картина. Из 256 выполненных базовым алгоритмом циклов в 100 случаях допустимых грамматик не оказалось, а в 156 других была построена одна из 6 вышеуказанных грамматик. Диаграмма распределения полученных результатов по значениям параметров t и h приведена на рис. 16, где каждая из 256 точек окрашена в один из 7 цветов, соответствующих результату исхода обработки образца при значениях t и h, отвечающих данной точке.

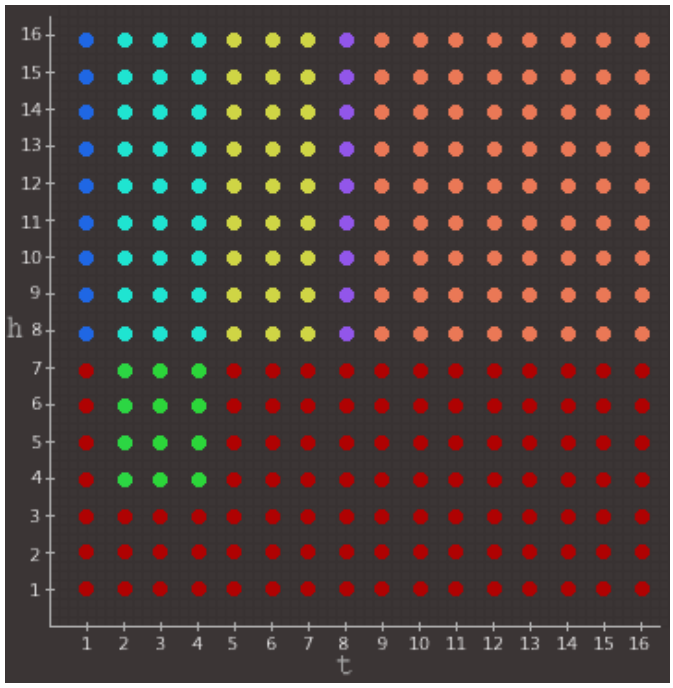

Рис. 16. Диаграмма распределения грамматик в целочисленных координатах t и h.

IV. Заключение

В данной работе доказана реализуемость базового алгоритма восстановления всевозможных допустимых грамматик для заданного положительного образца, т. е. при всех допустимых значениях параметров t и h.

Задействованные в базовом алгоритме параметры *h* и *t* представляют собой численные характеристики КС-языка, породившего предложения из Ω. Существование и смысл *h* и *t* вытекают из общих свойств КС-языков:

1. для любого КС-языка существует порождающая его

КС-грамматика специального вида [5], в которой:

- *для любой пары нетерминалов найдется предложение длины ≤ t, выводимое из одного нетерминала и не выводимое из другого;*

2. для любой КС-грамматики специального вида можно построить конечную сеть,

- *достаточную для восстановления грамматики [9]*

и

- *позволяющую вычислить параметр h как максимальное расстояние от входного полюса сети до её p-подсетей.*

Базовый алгоритм строит так называемые разделенные грамматики [2], в которых правые части правил, относящихся к одному нетерминалу, начинаются различными терминальными символами. Установлено, что для любой разделенной грамматики $G_{\text{И}}$ существуют значения параметров *f*, *h* и *t*, при которых базовый метод восстановления, примененный к $\Omega[f]$, строит разделенную-грамматику-специального-вида G эквивалентную $G_{\text{И}}$. При этом

– по определению эквивалентности: $L(G) = L(G_{\text{И}})$;

и $\Omega[f]$ удовлетворяет условию:

$\{ \text{длина(x)} \le f \mid x \in L(G_{И}) \} \subseteq \Omega[f] \subseteq L(G_{И})$.

В реальных задачах рассчитывать на знание параметров *h* и *t* особенно не приходится, можно лишь утверждать, что их значения ограничены максимальной длиной заданных предложений. А поэтому многократное применение базового алгоритма к одному и тому же набору предложений (при разных допустимых *h* и *t*) порождает ограниченный набор разделенных грамматик, среди которых найдется одна эквивалентная $G_{\text{И}}$.

При ближайшем рассмотрении выясняется, что базовый алгоритм допускает большое количество модификаций и улучшений, позволяющих наилучшим образом настроить его на конкретную область применения.

V. Библиография

[1] Соловьев С.Ю., Базовый алгоритм восстановления разделенных грамматик. // Труды XIII национальной конференции по искусственному интеллекту с международным участием КИИ-2012, том 1. Белгород: Изд-во БГТУ, 2012. С. 209-218.http://www.park.glossary.ru/serios/texts/read_19.pdf

[2] Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции, том 1. - М.: Мир, 1978.

[3] Ахо А., Хопкрофт Д., Ульман Дж. Структуры данных и алгоритмы. - М.: ИД "Вильямс", 2003.

[4] Соловьев С.Ю. Эквивалентные преобразования контекстно-свободных грамматик // Информационные процессы, том 10, No. 3, 2010. с. 292-302.

[5] Соловьев С.Ю. Структура контекстно-свободных языков // Информационные процессы, том 11, No. 1, 2011. с.161-178.

[6] Яблонский С.В. Введение в дискретную математику. - М.: Наука, 1986.

[7] Соловьев С.Ю. Нормализация контекстно-свободных грамматик для целей грамматического вывода. // XII национальная конференция по искусственному интеллекту с международным участием КИИ-2010. Труды конференции. - М.: Физматлит, 2010, том 1, стр.218-224.

[8] Смит Б. Методы и алгоритмы вычислений на строках. - М.: Вильямс, 2006.

[9] Соловьев С.Ю. Подход к восстановлению контекстно-свободных языков. - В кн.: Автоматизация производства пакетов прикладных программ. - Таллин: Изд-во Таллинского политехнического ин-та, 1980, с.126-129.

[10] Изданные материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем» (OSTIS-2012), ISBN 978-985-488-683-1

[11] Изданные материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем» (OSTIS-2013), ISBN 978-985-488-956-6

Implementation of a basic algorithm of separated grammar inference

K.V. Saveliev

Abstract — The description of different human activity processes is performed using a set of knowledge relative to the specified process. This set is called also "a set of sentences". At the same time, this set of sentences by itself does not contain the knowledge of processes. It contains the set of rules this set of sentences is inferred - so called "grammar". Our interest is the further research of this grammar. So, it is the problem of grammar inference from a set of sentences. This paper presents the implementation of a basic algorithm of separated grammar inference, proposed by prof. S.Y. Soloviev.

Keywords - grammar inference; prefix graphs; separated grammar.